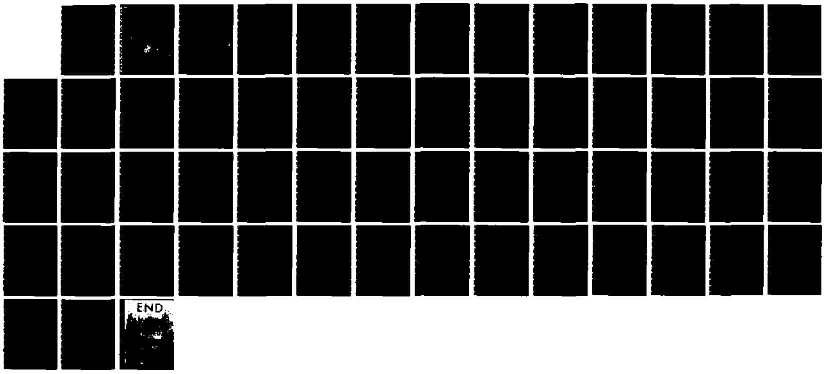
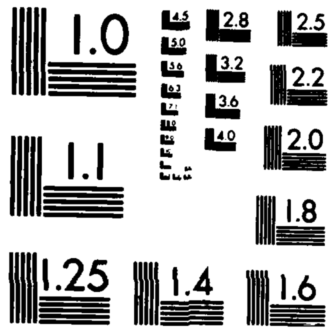
AD-A136 945 DIALOGUE MANAGEMENT: NEW CONCEPTS IN HUMAN-COMPUTER INTERFACE DEVELOPMENT..(U) VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG COMPUTER S.. 1/1

UNCLASSIFIED H R HARTSON ET AL. NOV 83 CSIE-83-13 F/G 5/8 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

DIALOGUE MANAGEMENT: NEW CONCEPTS IN HUMAN-COMPUTER INTERFACE DEVELOPMENT

H. Rex Hartson
Deborah H. Johnson

Virginia Polytechnic Institute and State University
Computer Science
Industrial Engineering and Operations Research
BLACKSBURG, VIRGINIA 24061

CSIE-83-13 November 1983

(1)

DIALOGUE MANAGEMENT:
NEW CONCEPTS IN HUMAN-COMPUTER INTERFACE DEVELOPMENT

H. Rex Hartson
Deborah H. Johnson

TECHNICAL REPORT

Prepared for
Engineering Psychology Group, Office of Naval Research
ONR Contract Number N00014-81-K-0143
Work Unit Number NR SRO-101

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSIE-83-13	2. GOVT ACCESSION NO. AD-A136945	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIALOGUE MANAGEMENT: NEW CONCEPTS IN HUMAN-COMPUTER INTERFACE DEVELOPMENT		5. TYPE OF REPORT & PERIOD COVERED Technical 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) H. Rex Hartson Deborah H. Johnson		8. CONTRACT OR GRANT NUMBER(s) N00014-81-K-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Virginia Polytechnic Institute & State University Blacksburg, VA 24061		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N42; RR04209; RR0420901; NR SRO-101
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, Code 442 800 North Quincy Street Arlington, VA 22217		12. REPORT DATE November 1983 13. NUMBER OF PAGES 51
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

design, human factors, languages, dialogue management, human-computer interface dialogue author, dialogue independence, internal dialogue, external dialogue

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Dialogue Management is an emerging field which emphasizes a specialization in the development of quality human-computer interfaces. It encompasses the design, implementation, simulation, execution, maintenance, and metering of dialogues in an integrated environment. Several key concepts in dialogue management have been identified in response to the need for improved human-computer interfaces. These concepts are surveyed here, and their importance to dialogue design and management is discussed. Dialogue independence and internal and external dialogue are manifest in the separation of the dialogue

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

components of a software system from the computational components. In a new system design role, a dialogue author is responsible for creating the dialogue which constitutes the human-computer interface of an application system A holistic methodological approach to system development places emphasis on the development of both dialogue and computational components of an application system. Systems are now being built which incorporate many of these important concepts in the management of dialogues for human-computer systems. Several of these systems are mentioned as examples of concepts application, and one such system is described in some detail.

A-1

S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The authors gratefully acknowledge the contributions to this research by other members in our research team, especially Dr. Roger W. Ehrich and Tamer Yunten. Dr. Ehrich was helpful in bringing to our attention the view of dialogue as a formal language problem in the DMS environment.

This research was supported by the Office of Naval Research under contract number N00014-81-K-0413 and work unit number NR-SRO-101. The effort was supported by the Engineering Psychology Group, Office of Naval Research under the technical direction of Dr. John J. O'Hare. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ABSTRACT

Dialogue Management is an emerging field which emphasizes a specialization in the development of quality human-computer interfaces. It encompasses the design, implementation, simulation, execution, maintenance, and metering of dialogues in an integrated environment. Several key concepts in dialogue management have been identified in response to the need for improved human-computer interfaces. These concepts are surveyed here, and their importance to dialogue design and management is discussed. Dialogue independence and internal and external dialogue are manifest in the separation of the dialogue components of a software system from the computational components. In a new system design role, a dialogue author is responsible for creating the dialogue which constitutes the human-computer interface of an application system. A holistic methodological approach to system development places emphasis on the development of both dialogue and computational components of an application system. Systems are now being built which incorporate many of these important concepts in the management of dialogues for human-computer systems. Several of these systems are mentioned as examples of concepts application, and one such system is described in some detail.

Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications--languages, methodologies, tools; D.2.2 [Software Engineering]: Tools and Techniques--modules and interfaces; structured programming; top-down programming; user interfaces; D.2.6 [Software Engineering]: Programming Environments; D.2.9 [Software Engineering]: Management--life cycle; D.3.2 [Programming Languages]: Language Classifications; H.1.2 [Models and Principles]: User/Machine Systems--human factors

General Terms: Design, Human Factors, Languages

Additional Keywords and Phrases: Dialogue management, human-computer interface, dialogue author, dialogue independence, internal dialogue, external dialogue

Dialogue Management:

New Concepts in Human-Computer Interface Development

H. Rex Hartson

Deborah H. Johnson

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, VA. 24061

"As we grow more familiar with the intelligent environment, and learn to converse with it from the time we leave the cradle, we will begin to use computers with a grace and naturalness that is hard for us to imagine today. And they will help all of us--not just a few 'super-technocrats'--to think more deeply about ourselves and the world." [TOFF80]

INTRODUCTION

As the "Gestalt of the computer" [ROSE74] becomes more pervasive in our society, the key to the real effectiveness of computers in that society will be usability by people other than computer professionals. As the quotation above suggests, the effectiveness of this amazing machine is limited not by its power to compute, but rather by its power to communicate. The roots of poor human-computer communication are deeply embedded in the methods currently used to design large interactive systems. Despite recent advances in methodological approaches to software design, these improvements have not yet addressed the important parallel issue of human-computer dialogue development. Many researchers have proposed viable solutions to specific issues in this area. Yet many of these issues represent only a small portion of a sizable problem. These existing issues and their solutions have generally been addressed without a framework for a larger strategy for dialogue management. It is the purpose of this paper to identify and assemble some concepts of this new field, to provide the beginnings of such a framework. In addition, many of the issues of human-computer interface development bear relationships to existing areas of computer science, particularly structured programming, software engineering, formal languages, and data abstraction. The key to a solution lies in reassessing the entire software development process. This paper reports on recent progress in the evolution of some new concepts, new roles, and new tools that are just emerging from this reassessment. Although artificial intelligence considerations such as natural language and knowledge-based user models are very much a part of the broader subject of human-computer dialogue, they are excluded from the scope of this paper.

Traditional Systems and Their Problems

Traditional software systems design is often aimed mainly at development of efficient and functionally correct computational code. Since getting the system debugged and operational depends mainly on the correctness of the computational component of the software, the effectiveness of the dialogue component gets little consideration.

An application programmer rarely has either the skills and knowledge or the time and patience to create effective, human-factored dialogues, but is typically much more intent upon the development of algorithms and other computational considerations. The software designer may even find it distracting, in the middle of a lengthy, logically complex piece of software, to have to write dialogue to communicate with the user. But this dialogue comprises the all-important interface with which human users must interact. If that interface is not created so that most humans can perform the task they set out to do, it matters little whether the computational part of the code works properly or not. Also in traditional systems, since both computational code and dialogue generally are written by an application programmer and are therefore tightly interspersed, the dialogue is actually a part of the computational software. This often makes it difficult to change easily either the dialogue or the computational part. The dialogue to be altered must first be found in a maze of functional code and then the desired changes made. When the dialogue is embedded in the computational code, changes to dialogue may propagate all the problems associated with changes to the computational program.

Related Work

Much research has been done in an attempt to improve human-computer communication. The groundwork for development of effective human-computer interfaces was laid during the last decade. This work was largely a collection of subjective "principles" for interface design [HANS71, MART73, KENN74, ROUS75, CHER76, MILL77, SMIT80]. Not surprisingly, among the leaders of this early work were some who specialized in graphics [FOLE74, NEWM79]. Very few of these principles are experimentally validated, however. Of the approximately 500 guidelines compiled from this and other relevant literature [WILL81], only a small portion had any empirical basis. Also, many c tradictions and inconsistencies were found among the "principle in the literature.

Only recently has empirical evaluation of the huma puter interface and its design been stressed [SHNE80] and communication identified as a key concept [HAYE79, THOM81]. Some researchers see the problem as one of dialogue [BLAC77, HART79, BARR80, MEHL81]. Research groups exist solely to give very strong emphasis to dynamic communication of information through the use of computerized tools [KAYA77, BYTE81]. Entire systems that emphasize development of the human-computer interface are now being created [FELD82, BORU82, HART82].

1. DIALOGUE MANAGEMENT: AN EMERGING FIELD

The discipline of dealing with the creation, modification, simulation, execution, testing, and metering of dialogues in an integrated manner has been called "dialogue management" [EHRI81]. The term is indicative of an emerging field that embodies a specialization in the development of quality human-computer interfaces.

1.1. FUNCTIONS OF HUMAN-COMPUTER DIALOGUE

Dialogue, as it relates to a human-computer interface in the traditional sense, refers to the interaction between a human and the computer system being used. It is the means by which the human gives commands or queries to the computer and by which the computer responds to or queries the user. This discourse is thus the exchange of words, phrases, parameterized commands, and other symbols and actions, i.e., the conversation, between a human and a computer.

This human-computer dialogue is composed of two separate parts: the computer's part and the human's part. The computer's part of the dialogue, of course, is actually written by humans and, therefore, could be considered to be human-to-human dialogue, offset in time. The human's part is any input a user may give to the system at execution time. It is confusing and indeed erroneous to think of a human-computer dialogue as being only what is actually displayed to the user by the system, (i.e., what is coded into the software itself). Instead, both the human side and the computer side of the dialogue must be considered in the design of a specific discourse. Multi-party grammars [SHNE82b] have evolved as a method to explicitly specify both the human input and the computer response for a human-computer dialogue.

Human-computer dialogue has two distinct functions: 1) to request information (either from the user or from the computer) and 2) to transmit information (either to the user or to the computer). Most programs, in order to execute, must have input (e.g., "name", "SSN", "edit", numerical data) from the user. This input is typically obtained through prompts or queries which ask the user for specific information. These requests are coded at appropriate points into the computational software itself. When the system needs to transmit information to the user (e.g., "searching database", "unmatched left parenthesis"), this is also done through some predetermined dialogue built into the software. A combination of transmitting and requesting information is manifest in a menu, which displays possible options and then asks the user to choose one. The content and format, as well as the logical sequencing, of dialogue is extremely important in determining how well the user can understand and manipulate the system.

1.2. NEW CONCEPTS IN DIALOGUE MANAGEMENT

As research has progressed, some concepts have emerged upon which a new approach to human-computer interface design can be based. It is the purpose of this paper to identify and highlight those concepts, primary among which are a software design and implementation concept called dialogue independence, a separate design role called a dialogue author, and a holistic methodological approach to interactive system design. The remainder of this paper will present these new concepts in the context of various current research efforts and will conclude by relating them to a particular system for dialogue management (DMS) which is being developed and implemented.

2. DIALOGUE INDEPENDENCE

2.1. WHAT IS DIALOGUE INDEPENDENCE?

The design of a large software system represents a significant commitment of resources in terms of both people and money. In traditional software systems, dialogue and computational code are inextricably interwoven together so that a commitment to one is a commitment to both. In the design of a system, major emphasis is placed on the development of the computational component. The result is that dialogue becomes less and less changeable as the design progresses. It is often later in the design process when the dialogue issues are understood more completely and human factors experiments can be performed, if necessary, to design a better human-computer dialogue. Unfortunately, by this time in the development process, the design is already committed to an ineffective dialogue.

Therefore, an important criterion for the design of human-computer interfaces is fast, easy modification of the dialogue of a specific system. Because research in this area properly involves empirical studies of human-computer interaction and dialogue design, this flexibility is even more important. The dialogue may need revision both for the experiments themselves, as well as to incorporate the results of experimentation back into the dialogues.

Several years ago, database researchers and designers encountered a similar problem in the need for easy modification of data without the necessity to change the corresponding programs. Data independence has emerged as a solution to this key issue in database development. Data independence is a concept that directs database design in such a way that changes to data are independent from, and therefore do not usually necessitate changes to, the application program which manipulates that data. A formal data definition allows the decoupling of data instances from the programs. An analogous concept, called dialogue independence [ROAC82a], allows changes to dialogue that do not mandate modifications to computational code.

Typically, a structured software system is divided into modules, which apply a single action or function to a single object. In traditional software systems, dialogue and computation are usually mixed together, often in the same module. This means that the need for a change to the dialogue may require searching through countless pages of source code to find the lines that need to be changed. This program structure can also make the process of following a program's computational logic very difficult, when READs and WRITEs to request and transmit and check and validate and re-request information between the user and the system are constantly interrupting the natural flow of the computational code. Dialogue independence helps to alleviate some of these difficulties by allowing easy alteration of dialogues without the associated problems of changing computational code.

Dialogue independence is achieved by both logical and physical separation of dialogue from computational software, which requires that dialogue and computational functions do not appear in the same module. This means that the executable code of an application system is composed of dialogue modules as well as computational modules. Intuitively, a dialogue module is used to carry out "one interaction's worth" of dialogue, and such modular packaging is similar to that commonly used for computational software. The "dialogue component" of a software system is a collection of all dialogue modules and the "computational component" is a collection of all computational modules. These components are bound together at some time before or during execution. As a result, dialogue and computational components can be modified to a great extent without each affecting the other.

Since there can be more than one dialogue component for a single computational component, an application system can have two or more very different user interfaces. Foley [FOLE81] and Feldman [FELD82] have captured this concept in their Abstract Interaction Handler, which contains knowledge of interaction styles, allowing their style-independent applications to be used with more than one kind of interface. Feldman and Rogers [FELD82] advocate the separation of the user front-end from the system semantics, even to the point of being able to customize user interfaces for individual users [FELD81]. The dialogue independence of their system semantics allows experimental evaluation of various interfaces while holding the underlying computational system constant. However, use of libraries of "interaction handling modules" may have limited advantages in light of the goal of tailoring an interface for different user classes as well as different users. It would seem that each application system must still be analyzed, and its specific dialogue needs addressed. Even though a given interaction module has "known human-factors attributes", those attributes were tested for a specific kind of interaction in a specific system and may not be applicable in a different situation.

This ability to keep flexible the design of a user interface relates to another important concept in dialogue management, that it is often as important to make a system human-factorable as it is to make it human-factored. Dialogue independence is a prerequisite for human-factorability, especially when it is not always clear during the early stages of system design what is needed to make a system human-factored.

2.2. SEPARATION, BUT NOT DIALOGUE INDEPENDENCE

The separation of the user interface from the rest of the application software is not an entirely recent idea. The literature contains several descriptions of systems for which such a separation was attempted after the system was implemented. That is, generalized user interfaces were developed to be used as add-on front-ends to existing application systems. [BASS81] and [WRIG78] each report on front-ends which were implemented to provide easy-to-understand dialogue with which to pass system commands and parameters. In [BASS81], the interface for a statistical analysis package operated in a system command and a job control command environment, and applied to both batch and on-line jobs. Essentially one interface was adapted to this diversity of use in one application. In [WRIG78], the interface was custom-coded to a single, specific medical application. Both of these groups began with an existing application system which did not have an easily usable interface, and attempted to revise the human-computer dialogue to meet their needs. Thus, the systems achieved separation of dialogue and computation, but not true dialogue independence, since the approach was not generalizable and extensible to other systems, or even to other interfaces for the same systems. These approaches, also, did not consider the computational component in the overall design.

The work on a demographic database system reported in [EVAN82] is somewhat similar. Here an adaptable user interface provided more than one dialogue to an existing software system as needed to suit the varying requirements of different user communities. Separation of the interface from the rest of the system made this possible.

Hayes, Ball, and Reddy [HAYE81] refer to the independence of the user interface from the application program as "tool independence". Their concept of a quality interface is one that supports graceful interaction in the sense that human-to-human communication is graceful and robust. Graceful interaction thus goes well beyond the traditional "principles" of human-computer interaction of the variety mentioned in section 1 of this paper and into the realm of natural language understanding [HAYE79]. Because of the enormous difficulties of producing such an interface, they propose to amortize this effort by building an application-independent system that will serve as the user interface for many application systems. Application independence is achieved by having all knowledge of the application in a separate declarative database, rather than in the interface itself. Such an interface system will also provide a consistent user view across all the applications for which it is used. Intuitively, however, use of the "same" interface would seem to preclude the possibility of tailoring the user interface over a broad range of application programs and user types (e.g., naive secretary to experienced engineer). It would seem better to generate different interface instances for each application. Consistency, in addition to the flexibility of a customized interface, could be achieved by using a comprehensive development methodology and a common set of tools (in the context of system-building, not in the Hayes, Ball, and Reddy context) to generate these interfaces.

2.3. DEVICE INDEPENDENCE

No discussion of dialogue independence would be complete without reference to the related and more well-known concept of device independence. Device independence is used to shield the application programmer from the special detailed characteristics of the devices through which the application system communicates with its users and from the technical differences in those details from device to device [HILL81, EHRI82]. Figure 1 shows the relationship of dialogue independence to device independence.

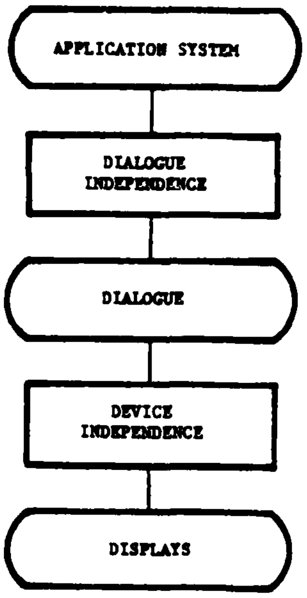


Figure 1. Dialogue Independence and Device Independence

2.4. INTERNAL AND EXTERNAL DIALOGUE

When an application system is separated into a computational component and a dialogue component, the computational part contains no mechanism for direct communication with the user; i.e., it contains no dialogue in the traditional sense, since all dialogue which interfaces with the user is now in the dialogue component. Under this kind of separation, an application system will consist, at its lowest level, of distinct dialogue modules and computational modules. The dialogue modules may contain the text and graphics which are displayed to the user, and where necessary, contain the mechanisms for receiving and processing user input and transmitting it to the computational part. The computational modules perform the functional processing. When the computational component needs information or data from the user to continue processing, it makes a call to the appropriate dialogue module. The dialogue module then interacts with the user to obtain the necessary input, possibly parses it, checks the input for type and range, and returns it to the computational module. Similarly, when the computational component needs to transmit information to the user, it again calls the appropriate dialogue module which displays the information to the user. Since the collection of computational modules thus interfaces with the user only through the dialogue modules, the computational modules would appear to be dia-

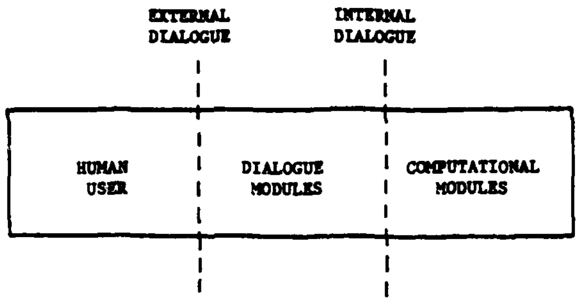


Figure 2. Internal and External Dialogues

logue-free. Computational modules, however, do engage in a less obvious form of dialogue, which is an "internal dialogue" with the dialogue modules so that the program can obtain data in order to execute. (See Figure 2.)

The interaction between the user and the dialogue modules is "external dialogue", the interaction which is typically thought of as making up the human-computer interface. External dialogue is highly varying in form and content. The part expressed to the user by the computer is limited only by the imagination of the person who creates the content of the dialogue modules and, hopefully, principles of good human-computer communication. Similarly, the possible inputs that a human can give to the computer are virtually infinite. Certainly only a very small subset of these possible inputs will be valid, but the potential for the others still exists and must be dealt with by appropriate messages.

Internal dialogue, on the other hand, has no direct connection to the user of the system, but serves as a link between the dialogue modules and the computational component of a system. It can therefore be formally specified, is much less variable in its form, and does not have to be human-understandable. The formal specification of internal dialogue is the real key to dialogue independence. Either dialogue or computation can be changed without affecting the other, as long as each remains consistent with the definition of their common interface.

These two kinds of dialogues can be seen to occur in most of the approaches which use separation of dialogue. For example, in [BORU82], external dialogue occurs at the user interface and internal dialogue happens at the application interface.

3. DIALOGUE AUTHOR

3.1. TRADITIONAL ROLES IN SYSTEM DESIGN

For many years, the two main roles involved in software development were those of the application programmer and the end-user of the system. These two types, however, frequently had severe communication problems. The programmer, impatient to begin coding, often had difficulty understanding the user's requirements. Similarly, the user was not often able to articulate the needs of the system and was baffled by the strange "computerese" in which the programmer tried to explain what was happening. The lack of communication between these two often resulted in systems that were not what the user wanted or needed, but what the programmer decided to provide. Gradually, the need was recognized for someone who could understand both the technical (programmer) and non-technical (user) sides of the system. This role is that of a systems analyst.

The user, however, might possibly be knowledgeable in only the one or two facets of the system used most. So the role of application expert evolved, one who knew all aspects of system requirements and uses thoroughly. The systems analyst serves as a liaison between application expert (and/or users) and application programmer, aiding and guiding both in defining and communicating system requirements. But neither the systems analyst nor the application expert was concerned primarily and explicitly with the human-computer interface. In the last few years, human-factors specialists have become an increasingly important part of computer system design and development teams for just this reason. They attempt to emphasize the incorporation of human-factors considerations into computer systems, and especially into the dialogues, so that the user has an effective interface with which to interact. But a human-factors expert frequently knows little about implementation of dialogues. Thus, a new role, that of a dialogue author, has been created to design and implement dialogues. The term "dialogue author" was first presented in [EHRI81] and rapidly began appearing in subsequent literature.

3.2. FUNCTIONS OF THE DIALOGUE AUTHOR

For the programmer, communication with the machine is very different from communication with a human. Thus, an application programmer, whose skill has been developed to deal with computers, is not a likely candidate to produce human-oriented dialogue. The knowledge and skills required to create effective human-computer interactions are quite different from those needed to write computational software. The programmer may even find it annoying in the midst of a logically complicated section of code to have to "change modes" and write dialogue to interact with the user, for example, to solicit an input value. It is even more distracting from the computational programming task to have to check the value received from the user for all possible errors and to respond, in case of an error, with consistently worded error messages and prompts for correct inputs. Thus, these interactions are frequently written as quickly as possible, with little thought given to their form or content.

Additionally, the independence of dialogue modules from computational modules indicates the need for separate roles to create the two components. Thus, the role of a dialogue author has been developed. The dialogue author is a skilled communication specialist whose ultimate goal is to create and implement human-factored dialogues. Because application programmers will no longer write dialogues, the dialogue author "will have sole responsibility for creating an interface for a user. The dialogue author is not specifically a programmer, but rather is a specially trained, human-factors-oriented professional concerned with the high level sequencing of computer-user interaction, as well as the form, style, and content of specific dialogues" [HART82].

In addition to creating human-factored dialogues, the dialogue author is interested in evaluation and revision of dialogue modules. Particularly since much human factors research encompasses empirical testing of dialogues, the dialogue author is interested in metering the dialogues and in incorporating newly discovered principles and guidelines into the dialogues. The separation of dialogue and computational components of the software through dialogue independence allows these modifications to be made quickly.

3.3. TOOLS

The people working in this relatively new field of human-computer interaction seem to be very tool-oriented. Many recent articles in the literature describe tools for the development and testing of human-computer dialogue. This is undoubtedly the "first wave" of an even more comprehensive phenomenon, related to the fact that people who are designing human-factored interfaces for others are not content to have poor quality interfaces for their own work. The dialogue author, even though creating a component of a software system, does not program the dialogue modules in the way that an application programmer programs the computational modules. Instead, the dialogue author requires a special environment designed specifically to aid in the implementation of dialogue modules.

Among the tools being developed to design, implement, simulate, test, and maintain dialogue are special functions to format text, generate graphics, produce keypad outlines and touch-panel configurations, and voice I/O managers. Generic tools that serve the dialogue author across the spectrum of these specific tools include dialogue editors, dialogue simulators, and dialogue databases. Rule-based "expert" dialogue designers [ROAC82b, FISC82] guide the author in applying human-factors considerations, graphic design principles, and guidelines for effective communication.

The tools generally fall into two categories: those that are used as dialogue design aids to help the author design the content and sequencing of dialogue modules (e.g., a design "expert" and a dialogue simulator) and those that are used as dialogue implementation aids to help the author produce dialogue modules with a particular content (e.g., text and graphics formatters and dialogue editors). There are also tools to facilitate experimentation with human factors in interactive displays [FOLE81, FELD81], including languages and metrics for interface specification and ergonomic evaluation

[BLES81], and tools to implement early and inexpensive experimental user interfaces [FELD82].

The TAXIS |MYLO80] system for designing and programming interactive information systems includes high-level programming language facilities integrated with database management and integrity checking mechanisms. TAXIS provides tools for the specification of the semantic (computational) component of interactive information systems. Barron [BARR80] has extended TAXIS tools to a pragmatic component for the description of processes that construct dialogues. Modified Petri nets are used to represent the organization and structure of dialogue and process control.

Borufka, Kuhlmann, and ten Hagen [BORU82] employ a mechanism called "dialogue cells" which provide a basic approach and several tools to aid in the programming of dialogues for interactive user interfaces. Separate roles include those of an application programmer and a dialogue designer. A dialogue tool providing input/output access to graphical devices reflects their heavy emphasis on graphics. Programming language extensions for programming of dialogues are part of their approach to dialogue construction, as an extension of the programming activity. Dialogue "cells" are proposed as a generic model for instances of human-computer interaction, using the basic elements of prompt, echo, user input symbol, and user input value. Dialogue cells are combined to model sequences of interactions.

3.4. INTERFACES AS LANGUAGES

The kind of tools discussed above, for formatting graphical and textual parts of dialogue, clearly should be included as part of a dialogue tool facility. This section singles out for discussion a less obvious tool area, but one which is perhaps the most important of all. Because the user of a human-computer application system must interact with that system through dialogue, it is evident that this dialogue comprises the interaction language between the system and its users. Beyond this apparent relationship, a view of the various interactions that occur through interfaces as expressions from formally defined languages can be useful in learning how to design the software that deals with dialogue.

For this purpose, user-computer interactions can be categorized into one of two classes: 1) unilateral command string or 2) request/response sequence. The unilateral command string variety of interaction begins with a prompt, and, therefore, itself is part of a generalized request/response sequence. But the prompt is usually a standard prompt not specific to the situation. For example, one might see a single "Ready" message at the command level of an operating system or text editor. A command in this case is "unilateral" in the sense that it is user-initiated and the system does not present to the user the choices for command names, followed by a series of interactions to obtain the parameters and options. The command is accepted, lexically analyzed, parsed, recognized as a valid command or as an error, and acknowledged as a valid command (sometimes implicitly through the presentation of the next prompt) or error. The action requested by the command, if valid, is performed. The commands are phrased in a formally defined grammar, and there is no difficulty in seeing this as a situation involving an interaction language whose syntactic form can be defined by a Backus-Naur Form (BNF) grammar or by any other commonly used, formal language-defining technique. Each command has a well-defined set of semantic actions.

The request/response case does not require the user to unilaterally synthesize a command, but allows the user's needs to be conveyed by a sequence of request/response interactions. The syntactic forms for request/response interactions include menus, touch panels, keypads, and ordinary question-and-answer text. The same kind of command that is found in the unilateral case can be built up by selecting a function and then selecting (or specifying), one at a time, the parameters and options. The user is coached for each item and does not need to know a syntax for the command. In a system without separation of dialogue and computation, the application software must handle these interactions by asking for each input, accepting it, performing lexical analysis (and sometimes parsing), recognizing the input as valid or erroneous (including validation of values for type and range), and then performing the required action. In a system with separation, the programmer need not be concerned with all of these functions, and the dialogue author can deal with them through a set of automated tools.

The unilateral command string is often considered to be most suited for "expert" users, while request/response interactions, are typically reserved for verbose "novice" users. At least one system, however, [ROBE79] uses menus within a fast communication system as its primary means of interaction with all levels of users.

While the syntactic forms of request/response sequences are quite different from those of regular command string languages, many of the processing requirements (i.e., lexical analysis, parsing, recognition, and validation) are similar. The difference is that the request/response approach breaks the application system down into numerous "micro-languages" whose processing can be done in a localized manner. That is, at a given point in a particular sequence, only a very small grammar is required to describe the valid inputs at that point.

Researchers have realized for some time that severe limitations exist in the notations typically used to specify a language. Representational schemes are generally so highly-coded that they become almost unreadable to the average person. Formal language definitions meet with resistance, especially from more practically oriented users, because they are so cryptic and often difficult to understand. The reader of a language definition should be able to quickly grasp the formalism and, from it, extract the information needed. This can be done only through the development of a readable notation that will describe the complete syntax of a language [LEDG74]. Attention to the design of the language specification, so that the general user can understand it, will help overcome some of the resistance normally encountered in the use of formal language definitions [MARC76].

There are several established methods of language syntax representation, the best known example of which is Backus-Naur Form (BNF)

[NAUR63]. However, most of these are used primarily for representation of static programming languages and are inadequate for representation of dynamic interaction languages [JACO82]. BNF has some human factors problems as well. It is a highly-structured, hierarchical metalanguage that results in a "fan-out" problem. That is, non-terminals in an expression can be replaced by more non-terminals through several successive iterations before a terminal symbol is finally reached. This multi-level tree structure is difficult to follow, since by the time the leaves (terminals) are reached, the root (highest level expression) may long be forgotten.

One variant of BNF, designed specifically to represent interactive languages rather than static languages, is the multi-party grammar [SHNE82b]. The features which differentiate this extension from standard BNF are the labeling of nonterminals with a party (i.e., either human or computer) identifier, assignment of values to nonterminals when appropriate, and definition of a nonterminal which will match any input string if no other parse of that input is successful. Other issues involving visual features peculiar to interactive displays are also incorporated. Because of their interactive nature, request/response sequences are good candidates for description by multi-party grammars [SHNE82b]. The idea of describing interfaces by a formal language specification has also been addressed by Foley [FOLE81]. A rigorous definition of input, output, and the relationship between them leads to a more structured design environment and again emphasizes human-computer interface development. Formal grammatical descriptions, combined with experimental results, are being used to compare alternative human-computer interface designs for ease of use [REIS81].

A fundamental basis underlying these methods for language representation is the notion of a metalanguage, which is a language to describe the syntax of other languages. All of the approaches to language specification described above contain examples of metalanguages. One of the great difficulties in dealing with metalanguages through a computer interface is in keeping the symbols of the metalanguage disjoint from the symbols of the language being specified. Since most user languages make use of all the keys on a typical terminal keyboard, there are none left to use unambiguously as metalanguage symbols. A simple example, but one that recurringly causes problems with naive users, is seen in the following informal language definition to a user:

Type "EXIT" to leave the program.

It is not clear (and cannot be made clear without special meta-metalanguage conventions concerning the use of quotation marks and upper and lower case letters) what the user should type. Should the quotation marks be typed? Must EXIT be typed in only upper case letters, or will lower case work, too? Other examples include the choice of symbols for delimiters (which can also be dependent on the context of that which is being delimited), the representation of required versus optional elements, notation for a choice among alternatives, special representation of conditionally optional components (including the conditions themselves), constraints on alternatives caused by choices made elsewhere in the expression, and limitations on values and characters to be used in parts of the command string.

Because of this strong relationship of human-computer interaction to language construction, consideration of language design and implementation must be a part of dialogue management. Automated tools are needed for both language specification and definition, and parser generation for language recognition. Since the dialogue author is not expected to be a language specialist, human factors is an especially important part of those tools.

3.5. DIRECT MANIPULATION

Direct manipulation is a concept relating to interfaces as languages. Direct manipulation allows a user to move the cursor to objects (both graphical and textual), move objects, and perform operations on objects with immediate visual feedback. Through direct manipulation [SHNE82a], naive users can use computing facilities without having to know a complicated command language. The "syntax" has become less computer-oriented and more user-oriented. Computer arcade games are striking examples; it is impossible to imagine playing Tempest by typing in the kind of commands one typically finds in a line editor!

System design for direct manipulation requires the means for an exact visual representation of the objects and their relationships. This requirement reflects a concept in dialogue management sometimes known as: "What you see is what you get". It is antithetical to this concept, for example, to build a file of text with imbedded commands for its formatting, as one does with many present-day text formatters. A facility for visual programming [MACD82] ought to be offered in the tools provided to system designers and builders, especially to the dialogue author. It is more direct for an author to build a display for an end-user by creating and manipulating boxes, circles, and text on a display screen than to build it by dialogue programming with an extended programming language, as in [BORU82].

4. HOLISTIC METHODOLOGICAL APPROACH

As the preceding section has shown, the importance of a structured approach to software development is well-recognized. However, the addition of the dialogue author to the overall system design team increases communication complexity in an already sizable group (including application programmer, systems analyst, application expert, user, and human factors expert). One of the most important concepts in dialogue management relates to the need for a system design methodology which emphasizes a holistic approach to software design, so that both dialogue and computational components of a system are given consideration. Such a methodology must also facilitate inter-role communication between the dialogue author and the application programmer.

A holistic approach to design methodology treats dialogue design as an integral part of the entire software system design. Without this integration, one can perhaps achieve separation of dialogue from application programs, but not true dialogue independence. This is why tuning the dialogue with an add-on front-end to an existing application is not entirely effective. "Silk purse" user interfaces cannot be fabricated from "sow's ears" logical control structures, no matter how well the dialogue form and content are refined.

The development of a large software system is a complex task even without considering the necessity for a human-factored end-user interface. Thus, the relatively new field of software engineering has devoted much of its effort to reducing this development complexity with little regard for the human factors that so greatly affect the performance of the end-user of the system. Current software development methodologies are aimed at facilitating the job of the application programmer by providing tools and guidelines for system analysis, design, implementation, and maintenance. They rarely consider the guidelines, methods, and tools needed to produce quality human-computer interfaces. As a result of using these methodologies, the effectiveness of the human-computer interface varies according to the human factors knowledge of the application programmer, and with the emphasis that the programmer gives to this interface. Recent advances in methodological development have seen more emphasis on the role of the human factors expert in system design. But just as the programmer's knowledge of human factors is varying, the human factors expert's knowledge of programming is varying. Therefore, for these two specialists to work together effectively, a methodology is required which embodies the needs and principles of both the discipline of software engineering and of human factors engineering, while providing appropriate development and communication tools for each.

Creating an application system which includes both dialogue and computational components is different from the development of a traditional software system. Traditional software methodology is inadequate because it fails to fulfill the following requirements:

1. Integration of human factors inputs from the beginning of system development,
2. Separation of dialogue author and programmer roles,
3. Minimization of computer knowledge required for system design,
4. Incorporation of both data flow and control flow in a single unified representation,
5. Production of requirements specifications that are easy to verify in the design, and
6. Use of an automated tool environment for development of dialogue and computational components.

An example of a holistic methodology which meets these requirements will be discussed in section 5.2.3 in the context of a specific system for dialogue management.

Foley [FOLE81] reports on a four-phase methodology for designing a user interface. The first step, conceptual design, specifies the user's view of the application system. The semantic design phase determines system functionality, while syntactic design deals with sequencing of input and output. Finally, the lexical phase maps syntax to specific hardware devices. This kind of methodological approach gives emphasis to development of the user interface, while providing the structure to handle the complexity of application system design.

5. A DIALOGUE MANAGEMENT SYSTEM

The concepts discussed in this paper represent a new approach to the management of dialogues in the development of human-computer interfaces. However, recognition of these concepts is not sufficient to produce improved human-computer interfaces. Demonstration of the effectiveness of these concepts requires application system development facilities and tools which incorporate them from the first phases of design all the way through to implementation. As one possible system to illustrate the concepts, a Dialogue Management System (DMS) [HART82] is described below.

5.1. OVERVIEW OF DMS

DMS is a comprehensive system for designing, analyzing, implementing, and monitoring human-factored interactive application systems. The high-level DMS organizational framework which supports the concepts of dialogue management is shown in Figure 3. DMS provides an environment which facilitates the design and implementation of both dialogue and computational components in parallel. The components of the environment which effect this parallel development are shown.

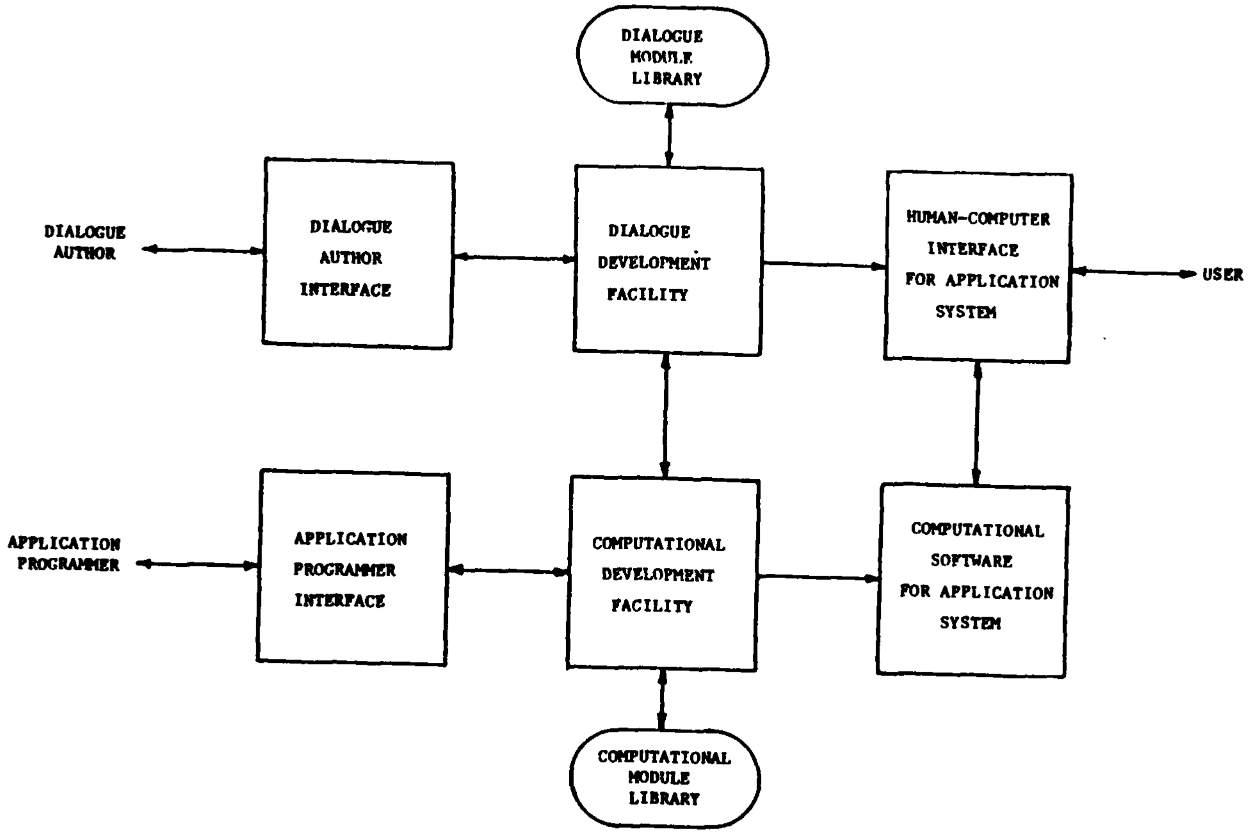

DIALOGUE MODULE LIBRARY
DIALOGUE AUTHOR
DIALOGUE AUTHOR INTERFACE
DIALOGUE DEVELOPMENT FACILITY
HUMAN-COMPUTER INTERFACE FOR APPLICATION SYSTEM
USER
APPLICATION PROGRAMMER
APPLICATION PROGRAMMER INTERFACE
COMPUTATIONAL DEVELOPMENT FACILITY
COMPUTATIONAL SOFTWARE FOR APPLICATION SYSTEM
COMPUTATIONAL MODULE LIBRARY

Figure 3. High-Level Organization of DMS

5.2. DIALOGUE MANAGEMENT CONCEPTS IN DMS

The concepts of dialogue management which have been presented in this paper are embodied in DMS. The implementation of dialogue independence, the role of the dialogue author, and a holistic methodological approach form the underlying basis of DMS.

5.2.1. Dialogue Independence

Dialogue independence is manifest logically in DMS through the use of separate modules for dialogue and computation in the application systems produced using DMS. Separate interfaces are provided for the creation of dialogue and for computational software. The integrated dialogue components and computational components form the completed application system. Physically, dialogue independence is achieved in DMS by the use of a multiprocess execution environment. In this environment, the dialogue component and the computational component are executed in separate, but concurrent, operating system processes. Finally, device independence is accomplished by generalized device drivers for each hardware device which is used in DMS (e.g., a graphics terminal and a black-and-white CRT), to localize the characteristics of, and differentiate among, display devices.

5.2.2. Dialogue Author

The role of the dialogue author is separate from that of the application programmer in DMS. As shown in the DMS structural organization, different interfaces and tools are provided for each role. The application programmer, through the application programmer interface, uses the computational development facility to create the computational component of an application system. No external dialogue, or interaction with the user, is produced by this portion of DMS. Only the internal dialogue, or the interaction between dialogue and computational components, is a part of the software developed using this tool. At the same time, the dialogue author uses the author's interface to create the dialogue portion of an application system. This Author's Interactive Dialogue Environment (AIDE) is the portion of DMS with which a dialogue author designs, implements, simulates, tests, and modifies dialogues. It provides an automated set of tools for a dialogue author to use in creating human-factored human-computer dialogues. This set of tools consists of such aids as a dialogue editor, a graphical formatter, a menu formatter, a voice I/O manager, a language implementer, a dialogue simulator, and a dialogue database. The output produced by AIDE is external dialogue, or the interface through which an end-user communicates with the completed application system. A complete presentation of AIDE, its tools, and its use by a dialogue author can be found in [JOHN82a].

A language implementer, one of the tools of AIDE, assists the dialogue author in the design, specification, and recognition of interaction languages [JOHN82b]. The language implementer consists of two main components: 1) a tool to aid in creating an interaction language and its specification for a specific application system, and 2) a tool to generate a parser to recognize this interaction language, once it is incorporated into the application system. Syntactic forms include menus, touch panels, keypads, and command strings. Additionally, the language implementer alleviates the difficult problem of determining whether the dialogue author or the application programmer should do input parsing. Input from the user to the system is really part of the external dialogue, and as such, should be handled by the dialogue author. But an author generally does not have the skills necessary to create parsers. The language implementer is an automated tool in AIDE that the dialogue author can easily use to do input recognition when needed, through the automatic generation of lexical analyzers and parsers.

Direct manipulation and "what you see is what you get" are achieved by having a very general display representation. Each screen in represented in the AIDE environment within a high performance (primary memory) relational Screen Database (SDB) containing relations to hold attribute values for the screen and its objects and groups of objects. The SDB is manipulated by a set of generalized editing functions. The direct manipulation functions (via keypad selection and cursor movement through the author's interface) use these editing functions to manipulate the objects on the screen and therefore in the database.

The display on the screen at the author's workstation is constantly kept current with the SDB representations by a Display Interpreter. Storing dialogue in an interpretable representation produced by AIDE tools is a key to dialogue independence, since local changes to dialogue do not affect the computational component of an application. Screen definitions are retrieved (stored) into (from) the SDB from (into) the Dialogue Database (DDB), which holds the dialogue for a complete application system. For performance reasons, special methods are applied to "compile" the DDB contents when the application is ready for execution.

5.2.3. Holistic Methodology

Because traditional software methodologies have been found to be inadequate for dialogue management, a new methodology, called the Supervisor-Based System Development Methodology (SBSDM) has been developed [YUNT82] for use in DMS. Applicable to the modelling of any procedural system, the SBSDM integrates many of the features of conventional methodologies (the equivalent of data flow diagrams, structure charts, PDLs, structured design, etc.) into a single approach, which integrates human factors considerations from the earliest stages of design. Based on a separation of dialogue author and application programmer roles, and the fact that the author need not be a computer specialist, the SBSDM combines both data flow and control flow into requirements and design representation at all levels of system development.

The basic element of the SBSDM is a "supervisory structure",

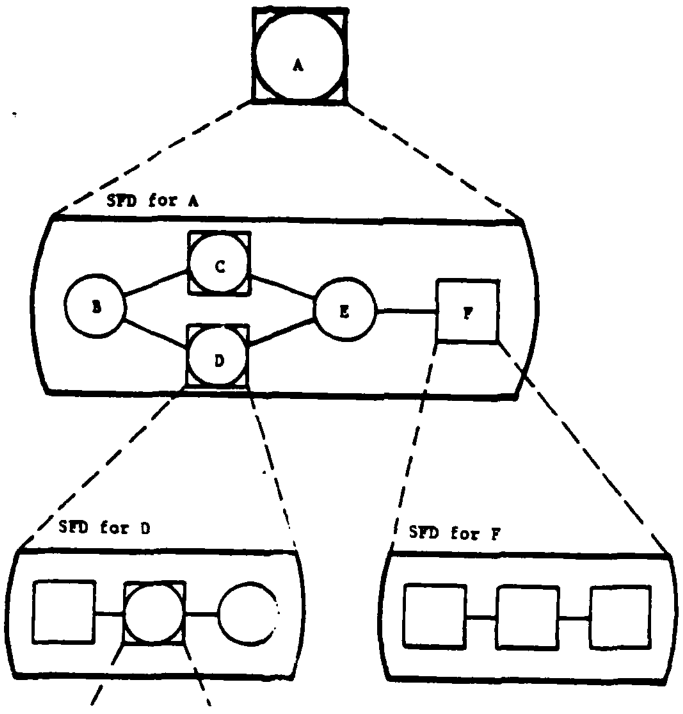


Figure 4. Supervisory Structure and Supervisory Cells

which is a hierarchy of "supervisory cells" (see Figure 4). Each cell represents the subfunctions of a single supervisory function. A Supervised Flow Diagram (SFD) represents the sequence of subfunctions and indicates both control flow and data flow among the subfunctions. The supervisory function of an SFD administers both data flow and control flow. Each subfunction can then be a supervisory function of supervisory cells at the next level in the hierarchy. Worker functions, which perform single dialogue or computational operations, are found in the terminal nodes of the supervisory structure. All the supervisory functions, taken together, make up the control structure of the entire application system. Details of the SBSDM, and examples of its use, are to be found in [YUNT82].

The supervisory structure is used to represent the application system in all phases of development in the software lifecycle. The requirements analysis and specification phase begins with a "wish list" from the application experts and users. This list is converted into a high-level supervisory structure by the human factors engineers and software engineers/systems analysts. When it has been expanded until its terminal nodes are pure computation or pure dialogue, this structure is called the "behavioral structure", which is a complete, precise statement of the system requirements. Only the development of the computational and dialogue supervisory functions in worker modules is required to complete the physical design. Implementation, then, amounts to a direct mapping of this representation to source code. This graphical SFD language of the behavioral structure is a very high-level programming language, and, since the methodology is supported by an automated tool environment, the source code for the entire control structure can be generated using a SFD language compiler. Thus, the behavioral structure provides a graphical representation that is itself executable. All application system sequences can be tested before any other code is written. Because the behavioral structure is executable, there is no need for a translation step from the requirements to the design and implementation, and no need to verify that the design and implementation meet the requirements. The only remaining tasks are the writing of the computational modules by the application programmer and the production of the terminal dialogue modules by the dialogue author.

This methodology recognizes and addresses the fact that in order to deal effectively with dialogue, system developers must work with the entire system. SBSDM incorporates the dialogue management concept of dialogue independence into system design, and provides for effective interaction between the new role of the dialogue author and the traditional role of the application programmer.

6. SUMMARY

The need for effective human-computer interfaces is well-recognized. However, most research in software engineering has emphasized mainly the computational component of a system, while the dialogue is developed as a secondary activity. In order to address this issue, several important concepts of dialogue management have been identified. Dialogue independence, which results in the separation of dialogue and computational components within a software system, allows easy modification of dialogues so that a human-computer interface can be quickly modified to meet the needs of its users. The role of a dialogue author, whose main purpose is to create dialogues that incorporate good human-computer interface guidelines, is another key concept. By automating as much of the dialogue development and implementation process as possible, the dialogue author is freed from much of the tedium of "coding" dialogues and can concentrate on incorporating human factors into the dialogues to create an effective human-computer interface. An Author's Interactive Dialogue Environment (AIDE) is being developed as one such system for use by the dialogue author. This environment consists of numerous automated tools for assisting in both dialogue design and implementation. This places the emphasis of human-computer interface development on what a dialogue will contain, not on how it will be implemented. Finally, a system which incorporates dialogue independence and which is created by a dialogue author and an application programmer working in parallel must be developed by employing a methodological approach which considers the entire software system. A holistic methodology of this type provides for coordination of the roles as well as integration of the dialogue and computational components which are independently produced.

Many aspects of human-factoring an application system user interface seem intuitively obvious, even to the casual observer. But these aspects are all too frequently ignored in the rush to produce an operational system as soon as possible. Anything that takes time away from programming is considered by many people to be non-productive. But application systems which are developed without proper attention to the human-computer interface are frequently ineffective for the end-user. With ever-increasing numbers of interactive computer users having wide ranges in experience, problems, and expectations, the need for effective human-computer interfaces becomes a primary input to system design. Ad hoc design methods and traditional manual development tools will rarely produce satisfactory systems. Providing a standard methodology and automated tools for the creation of useful, effective human-computer interfaces is "a challenge to scientific competence, engineering ingenuity, and artistic elegance" [SHNE80].

The concepts presented in this paper are being incorporated into an automated Dialogue Management System (DMS) to be used for the development of human-computer systems. However, such a DMS is not a prerequisite for improved dialogue design. More effective human-computer interfaces can presently be produced by using the concepts of dialogue independence, a dialogue author, and a comprehensive holistic methodology in the development of human-computer systems.

REFERENCES

BARR80 Barron, John L., "Dialogue Organization and Structure for Interactive Information Systems," Department of Computer Science Technical Report CSRG-108, University of Toronto (January 1980).

BASS81 Bass, Leonard J., and Ralph E. Bunker, "A Generalized User Interface for Applications Programs," *CACM*, 24, 12 (December 1981), 796-800.

BLAC77 Black, James L., "A General Purpose Dialogue Processor," *Proc. of the NCC*, (1977), 397-408.

BLES81 Bleser, T. P., "A Formal Language for Describing and Evaluating the Ergonomics of User-Computer Interfaces," *ACM DC Chapter 20th Symposium*, College Park, MD (June 1981).

BORU82 Borufka, H. G., H. W. Kuhlmann, and P. J. W. ten Hagen, "Dialogue Cells: A Method for Defining Interactions," *IEEE Computer Graphics and Applications*, (July 1982), 25-33.

BYTE81 *Byte: The Small Systems Journal*, Special Issue: Smalltalk, 6, 8 (1981).

CHER76 Cheriton, D. R., "Man-Machine Interface Design for Timesharing Systems," *Proc. of ACM Annual Conf.*, (1976), 362-380.

EHRI81 Ehrich, Roger W., and H. Rex Hartson, "DMS -- An Environment for Dialogue Management," *Proc. COMPCON81*, Washington, DC (September 1981), 121.

EVAN82 Evans, R., N. J. Fiddian, and W. A. Gray, "Adaptable User Interfaces for Portable, Interactive Computing Software Systems," *Proc. Part II, Conf. on Easier and More Productive Use of Computer Systems*, Ann Arbor, Michigan (May 1981), as in *SIGSOC Bulletin*, 13, 2-3 (January 1982), 59.

FELD81 Feldman, Michael B., "Tools to Facilitate Human-Factors Improvement in Interactive Information Display Systems," *Proc. COMPCON81*, Washington, DC (September 1981), 117-118.

FELD82 Feldman, Michael B., and George T. Rogers, "Toward the Design and Development of Style-Independent Interactive Systems," *Proc. Conf. Human Factors in Computer Systems*, Gaithersburg, MD (March 1982), 111-116.

FISC82 Fischer, Gerhard, "Symbiotic, Knowledge-Based Computer Support Systems," *Proc. of the IFAC Conf. on Analysis, Design, and Evaluation of Man-Machine Systems*, Baden-Baden, Germany (September 1982), 351-358.

FOLE74 Foley, James D., and Victor L. Wallace, "The Art of Natural Graphic Man-Machine Conversation," *Proc. of the IEEE*, 63, 4 (April 1974), 462-471.

FOLE81 Foley, James D., "Tools for the Designers of User Interfaces," George Washington University Institute for Information Science and Technology Report GWU-IIST-81-07 (March 1981).

HANS71 Hansen, W. J., "User Engineering Principles for Interactive Systems," *AFIPS Conf. Proc.*, 39, (1971), 523-532.

HART79 Hartson, H. Rex, and Michael D. Schnetzler, "Generalized Interactive User Interface Dialogue Designer," Working Paper (March 1979).

HART82 Hartson, H. Rex, and Roger W. Ehrich, "DMS: A System for the Management of Dialogue for User-Software Interfaces," submitted for publication (October 1982).

HAYE79 Hayes, Phil, and Raj Reddy, "An Anatomy of Graceful Interaction in Spoken and Written Man-Machine Communication," Carnegie-Mellon University Report CMU-CS-79-144 (September 1979).

HAYE81 Hayes, Phil, Eugene Ball, and Raj Reddy, "Breaking the Man-Machine Communication Barrier," *IEEE Computer*, (March 1981), 19-30.

HILL81 Hillsberg, Bruce L., "Generic Terminal Support," *ACM Operating Systems Review*, 15, 2 (April 1981), 10-15.

JACO82 Jacob, Robert J. K., "Using Formal Specifications in the Design of a Human-Computer Interface," *Proc. Conf. on Human Factors in Computing Systems*, Gaithersburg, MD (March 1982), 315-321.

JOHN82a Johnson, Deborah H., and H. Rex Hartson, "The Role and Tools of a Dialogue Author in Creating Human-Computer Interfaces," VPI&SU Departments of Computer Science and Industrial Engineering Technical Report CSIE-82-8 (May 1982).

JOHN82b Johnson, Deborah H., and H. Rex Hartson, "Interaction Language Specification and Recognition", VPI&SU Departments of Computer Science and Industrial Engineering Technical Report CSIE-82-11 (1982).

KAYA77 Kay, Alan and Adele Goldberg, "Personal Dynamic Media," *IEEE Computer*, (March 1977), 31-41.

KENN74 Kennedy, T. C. S., "The Design of Interactive Procedures for Man-Machine Communication," *International Journal of Man-Machine Studies*, 6 (1974), 309-334.

LEDG74 Ledgard, Henry, "Production Systems: Or Can We Do Better Than BNF?", *CACM*, 17, 2 (February 1974).

MACD82 MacDonald, Alan, "Visual Programming," *Datamation*, (October 1982), 132-140.

MARC76 Marcotty, M., Henry Ledgard, and G. Bochmann, "Sampler of Formal Definitions," *Computing Surveys*, 8, 2 (June 1976).

MART73 Martin, James, *Design of Man-Computer Dialogues*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973).

MEHL81 Mehlmann, Marilyn, *When People Use Computers*, Prentice-Hall, Englewood Cliffs, NJ (1981).

MILL77 Miller, Lance A., and John C. Thomas, Jr., "Behavioral Issues in the Use of Interactive Systems," *Int. J. Man-Machine Studies*, 9 (1977), 509-536.

MYLO80 Mylopoulos, John, Philip A. Bernstein, and Harry K. T. Wong, "A Language Facility for Designing Database-Intensive Applications," *ACM TODS*, 5, 2 (June 1980), 185-207.

NAUR63 Naur, P. (ed.), "Revised Report on the Algorithmic Language ALGOL 60," *CACM*, 6 (January 1963).

NEWM79 Newman, William M., and Robert F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, second edition, New York (1979).

REIS81 Reisner, Phyllis, "Formal Grammar and Human Factors Design of an Interactive Graphics System," *IEEE Trans. on Software Engineering*, SE-7, 2 (March 1981), 229-240.

ROAC82a Roach, John, H. Rex Hartson, Roger W. Ehrich, Tamer Yunten, and Deborah H. Johnson, "DMS: A Comprehensive System for Managing Human-Computer Dialogue," *Proc. Conf. on Human Factors in Computer Systems*, Gaithersburg, MD (March 1982), 102-105.

ROAC82b Roach, John, J. A. Pittman, Susan Reilly, and John Savarese, "A Visual Design Consultant," *IEEE Systems, Man, and Cybernetics Conference*, Seattle, WA (October 1982).

ROBE79 Robertson, G., D. McCracken, and A. Newell, "The ZOG Approach to Man-Machine Communication," Carnegie-Mellon University Department of Computer Science Report CMU-CS-79-148 (1979).

ROSE74 Rosenburg, Victor, "The Scientific Premises of Information Science," *J. of the ASIS*, (July-August 1974), 263-269.

ROUS75 Rouse, William B., "Design of Man-Computer Interfaces for On-Line Interactive Systems," *Proc. of the IEEE*, 63, 6 (June 1975), 847-857.

SHNE80 Shneiderman, Ben, *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, Inc., Cambridge, Mass. (1980).

SHNE82a Shneidermen, Ben, "Direct Manipulation: A Step Beyond Programming Languages," *Proc. Part II, Conf. on Easier and More Productive Use of Computer Systems*, Ann Arbor, Michigan (May 1981), as in *SIGSOC Bulletin*, 13, 2-3 (January 1982), 143.

SHNE82b Shneiderman, Ben, "Multi-Party Grammars and Related Features for Designing Interactive Systems," *IEEE Trans. on Systems, Man, and Cybernetics*, 12,2 (March-April 1982).

SMIT80 Smith, H. T., and T. R. G. Green, *Human Interaction with Computers*, Academic Press, London (1980).

THOM81 Thomas, John, and J. M. Carroll, "Human Factors in Communication," *IBM Sys. J.*, 20, 2 (1981), 237-263.

TOFF80 Toffler, Alvin, *The Third Wave*, Bantam Books, New York (1980).

WILL81 Williges, Beverly H., and Robert C. Williges, "User Considerations in Computer-Based Information Systems," VPI&SU Departments of Computer Science and Industrial Engineering Technical Report CSIE-81-2 (September 1981).

WRIG78 Wright, Peggy R., and Barry W. Brown, "A Processor for Providing Friendly Environments for Frequently Used Application Packages," *Proc. ACM Annual Conf.* 1, Washington DC (December 1978), 346-350.

YUNT83 Yunten, Tamer, and H. Rex Hartson, "A Supervisor-Based System Development Methodology," VPI&SU Departments of Computer Science and Industrial Engineering Technical Report CSIE-83 (1983).

OFFICE OF NAVAL RESEARCH

Engineering Psychology Group

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CAPT Paul R. Chatelier
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, DC 20301

Dr. Dennis Leedom
Office of the Deputy Under Secretary
of Defense (C^3I)
Pentagon
Washington, DC 20301

Department of the Navy

Engineering Psychology Group
Office of Naval Research
Code 442 EP
Arlington, VA 22217

Communication & Computer Technology
Programs
Code 240
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Physiology & Neuro Biology Programs
Code 441NB
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Tactical Development & Evaluation
Support Programs
Code 230
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Manpower, Personnel & Training
Programs
Code 270
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Information Sciences Division
Code 433
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine Corps
Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

CDR James Offutt, Officer-in-Charge
ONR Detachment
1030 East Green Street
Pasadena, CA 91106

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, DC 20375

Dr. Michael Melich
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, DC 20375

Dr. J. S. Lawson
Naval Electronic Systems Command
NELEX-06T
Washington, DC 20360

Department of the Navy

Dr. Robert E. Conley
Office of Chief of Naval Operations
Command and Control
OP-094H
Washington, DC 20350

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, DC 20350

Combat Control Systems Department
Code 35
Naval Underwater Systems Center
Newport, RI 02840

Human Factors Department
Code N-71
Naval Training Equipment Center
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis and Evaluation
Group
Orlando, FL 32813

CDR Norman E. Lane
Code N-7A
Naval Training Equipment Center
Orlando, FL 32813

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Mr. John Impagliazzo
Code 101
Naval Underwater Systems Center
Newport, RI 02840

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, DC 20380

Department of the Navy

Dr. L. Chmura
Naval Research Laboratory
Code 7592
Computer Sciences & Systems
Washington, DC 20375

Commanding Officer
MCTSSA
Marine Corps Base
Camp Pendleton, CA 92055

Chief, C^3 Division
Development Center
MCDEC
Quantico, VA 22134

Human Factors Technology Administrator
Office of Naval Technology
Code MAT 0722
800 N. Quincy Street
Arlington, VA 22217

Commander
Naval Air Systems Command
Human Factors Programs
NAVAIR 334A
Washington, DC 20361

Commander
Naval Air Systems Command
Crew Station Design
NAVAIR 5313
Washington, DC 20361

Mr. Philip Andrews
Naval Sea Systems Command
NAVSEA 03416
Washington, DC 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 81323
Washington, DC 20360

Larry Olmstead
Naval Surface Weapons Center
NSWC/DL
Code N-32
Dahlgren, VA 22448

Department of the Navy

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 06340

Commanding Officer
Naval Health Research Center
San Diego, CA 92152

Commander, Naval Air Force,
U.S. Pacific Fleet
ATTN: Dr. James McGrath
Naval Air Station, North Island
San Diego, CA 92135

Naval Personnel Research and
Development Center
Planning & Appraisal Division
San Diego, CA 92152

Dr. Robert Blanchard
Navy Personnel Research and
Development Center
Command and Support Systems
San Diego, CA 92152

CDR J. Funaro
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. Stephen Merriman
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. Jeffrey Grossman
Human Factors Branch
Code 3152
Naval Weapons Center
China Lake, CA 93555

Human Factors Engineering Branch
Code 1226
Pacific Missile Test Center
Point Mugu, CA 93042

Department of the Navy

Dean of the Academic Departments
U.S. Naval Academy
Annapolis, MD 21402

Dr. S. Schiflett
Human Factors Section
Systems Engineering Test
Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

CDR C. Hutchins
Code 55
Naval Postgraduate School
Monterey, CA 93940

Office of the Chief of Naval
Operations (OP-115)
Washington, DC 20350

Department of the Army

Mr. J. Barber
HQS, Department of the Army
DAPE-MBR
Washington, DC 20310

Department of the Navy

Dr. Edgar M. Johnson
Technical Director
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Director, Organizations and
Systems Research Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground, MD 21005

Department of the Air Force

U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, DC 20332

Department of the Air Force

AFHRL/LRS TDC
Attn: Susan Ewing
Wright-Patterson AFB, OH 45433

Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Dr. Earl Alluisi
Chief Scientist
AFHRL/CCN
Brooks Air Force Base, TX 78235

Foreign Addressees

Dr. Kenneth Gardner
Applied Psychology Unit
Admiralty Marine Technology
Establishment
Teddington, Middlesex TW11 OLN
England

Director, Human Factors Wing
Defence & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsview, Ontario M3M 3B9
Canada

Dr. A. D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, CB2 2EF England

Professor Brian Shackel
Department of Human Science
Loughborough University
Loughborough, Leics, LE11 3TU
England

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314

Other Government Agencies

Dr. Craig Fields
Director, System Sciences Office
Defense Advanced Research Projects
Agency
1400 Wilson Blvd.
Arlington, VA 22209

Dr. M. Montemerlo
Human Factors & Simulation
Technology, RTE-6
NASA HQS
Washington, DC 20546

Other Organizations

Dr. H. McI. Parsons
Human Resources Research Office
300 N. Washington Street
Alexandria, VA 22314

Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Dr. Deborah Boehm-Davis
General Electric Company
Information Systems Programs
1755 Jefferson Davis Highway
Arlington, VA 22202

Dr. James H. Howard, Jr.
Department of Psychology
Catholic University
Washington, DC 20064

Mr. Edward M. Connelly
Performance Measurement Associates, Inc.
410 Pine Street, S. E.
Suite 300
Vienna, VA 22180

Dr. Edward R. Jones
Chief, Human Factors Engineering
McDonnell-Douglas Astronautics Co.
St. Louis Division
Box 516
St. Louis, MO 63166

Other Organizations

Dr. Marvin Cohen
Decision Science Consortium
Suite 721
7700 Leesburg Pike
Falls Church, VA 22043

Dr. William B. Rouse
School of Industrial and Systems
Engineering
Georgia Institute of Technology
Atlanta, GA 30332

Dr. Richard Pew
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02238

Dr. Douglas Towne
University of Southern California
Behavioral Technology Laboratory
3716 S. Hope Street
Los Angeles, CA 90007

Psychological Documents
ATTN: Dr. J. G. Darley
N 565 Elliott Hall
University of Minnesota
Minneapolis, MN 55455

Mr. Richard Main
ONR Resident Representative
George Washington University
2110 G. Street, N.W.
Washington, DC 20037

Dr. E. Gloye
ONR Western Regional Office
1030 East Green Street
Pasadena, CA 91106

Dr. J. Hopson
HF Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. A. Meyrowitz
Code 433
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

Other Organizations

Dr. Thomas McAndrew
Code 32
Naval Undersea Systems Center
New London, CT 06320

Mr. Walter P. Warner
Code KOZ
Strategic Systems Department
Naval Surface Weapons Center
Dahlgren, VA 22448

Dr. Mel C. Moy
Code 302
Naval Personnel R&D Center
San Diego, CA 92152

Dr. Richard Neetz
Pacific Missile Test Center
Code 1226
Pt. Mugu, CA 93042

Mr. Rich Miller
NSWC
Code N32
Dahlgren, VA 22448

Dr. Arthur Fisk
ATT Long Lines
12th Floor
229 W. Seventh St.
Cincinnati, OH 45202

Dr. R. J. K. Jacob
Code 7590
Naval Research Laboratory
Washington, DC 20375